Universidade Federal de Alagoas (UFAL)
Campus Arapiraca

Programação Orientada a Objetos (POO)

12 - Padrão MVC e Padrão DAO

Alexandre de Andrade Barbosa

alexandre.barbosa@arapiraca.ufal.br

Objetivos

Objetivos

- Apresentar o padrão arquitetural MVC
- Apresentar o padrão de projeto DAO
- Descrever um exemplo ilustrativo da utilização do DAO

MVC

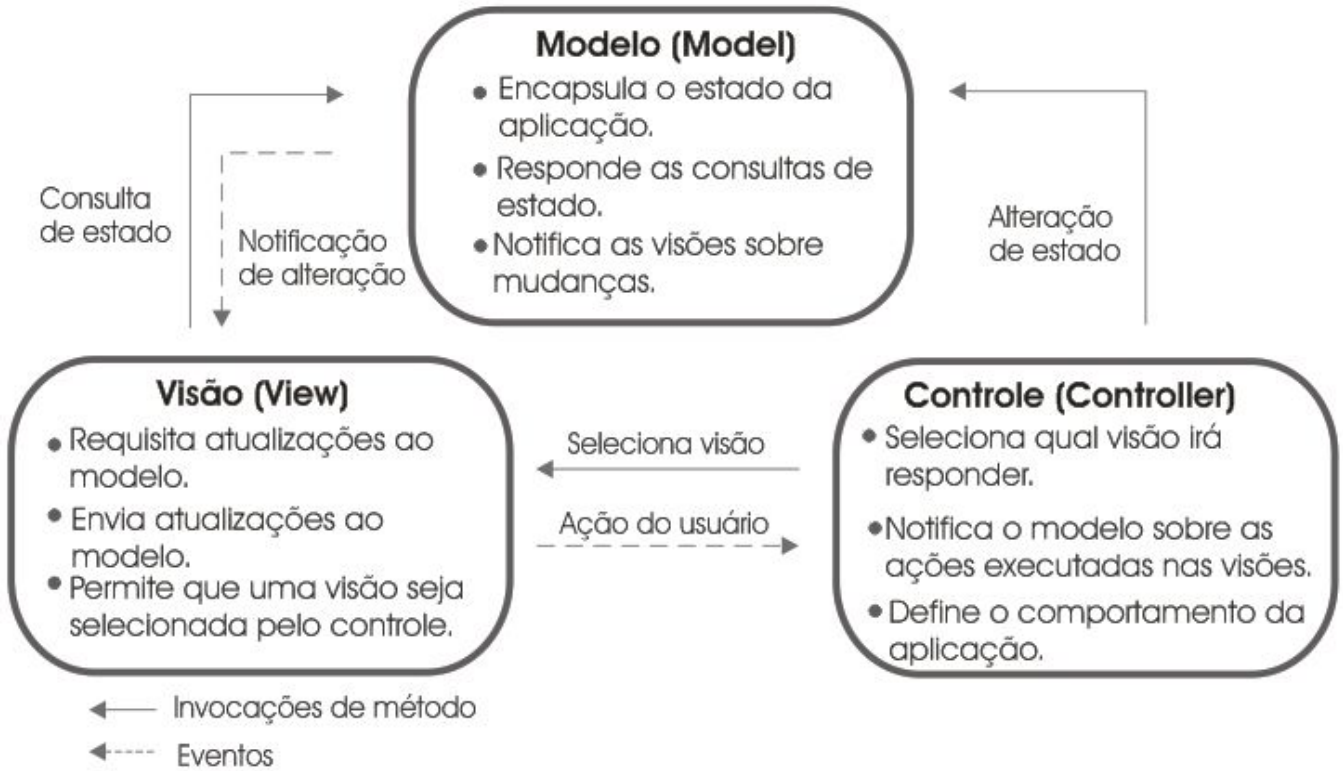
- MVC é um **padrão arquitetural**
- O software deve ser dividido em três partes interconectadas
  - Modelo (*Model*) - camada de manipulação dos dados
  - Visão (*View*) - camada de interação do usuário
  - Controle (*Controller*) - camada de controle

MVC

- Modelo (*Model*)
  - representação dos dados
  - leitura e escrita de dados
  - validações de dados
- Visão (*View*)
  - interação com o usuário
  - exibição dos dados
  - captura de dados
- Controle (*Controller*)
  - receber todas as requisições
  - camada de ligação entre visão e modelos

MVC
Padrões de projeto no MVC

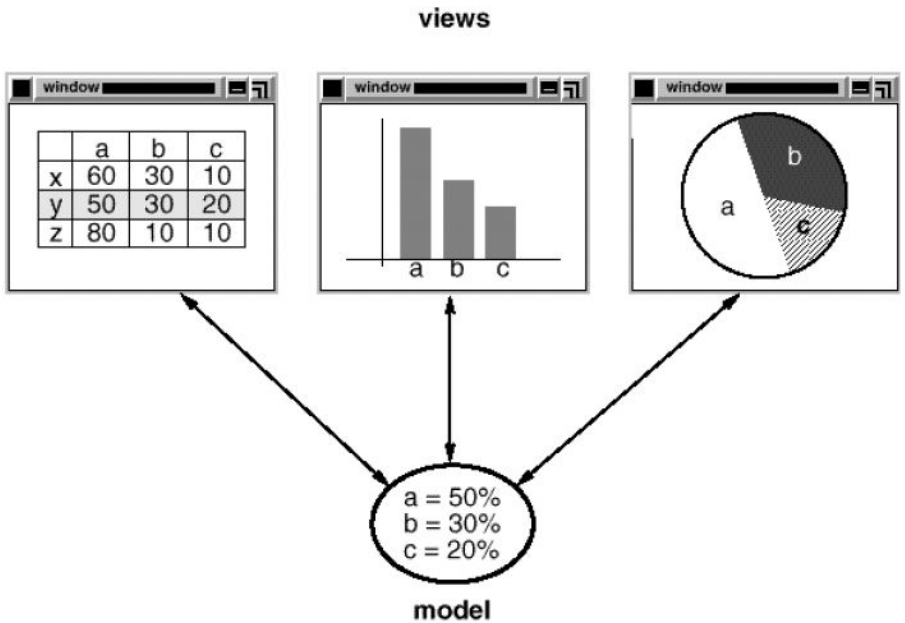
- Modelo-Visão-Controle (MVC)



MVC de Smalltalk

MVC
Padrões de projeto no MVC

- Uma visão deve garantir que sua representação reflete o estado do modelo
- Se o estado do modelo é modificado as visões associadas devem ser notificadas



Diferentes visões podem se relacionar a um único modelo

MVC
Padrões de projeto no MVC

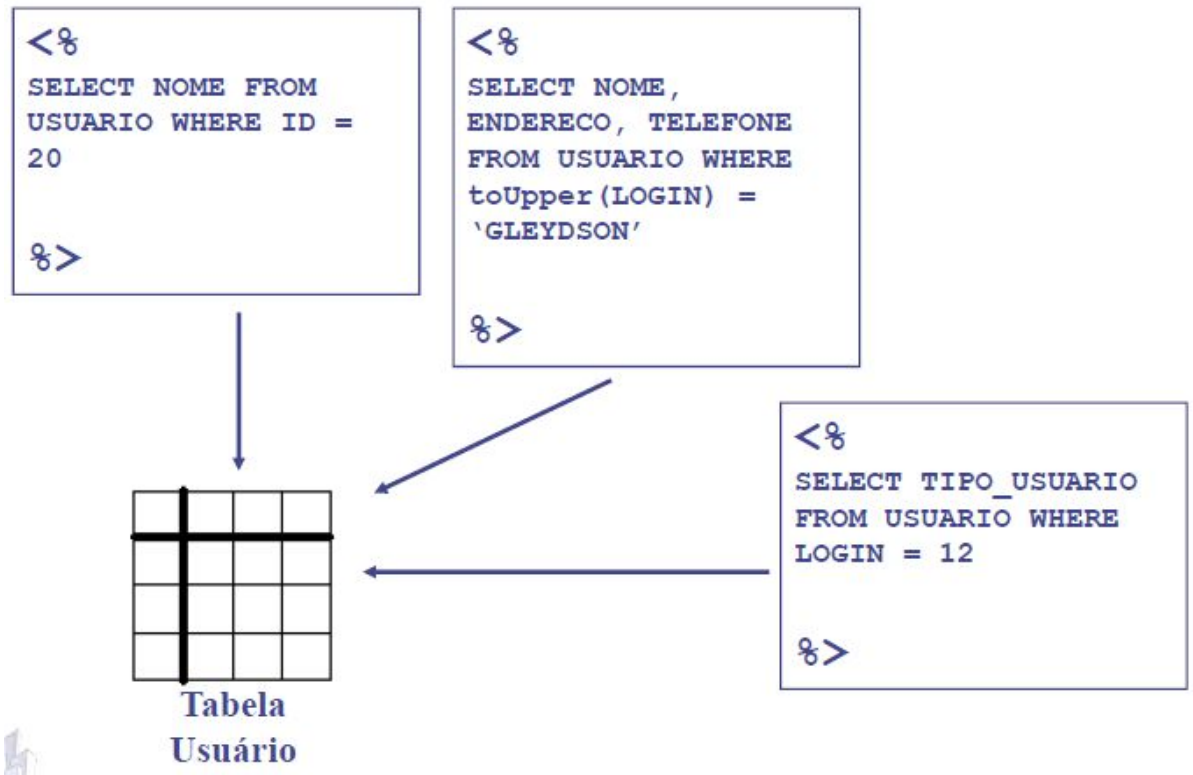
- Diferentes padrões de projeto são utilizados em MVC:
  - Observer - objetos que notificam mudanças de estado a outros observadores
  - Composite - objetos que são formados a partir de um conjunto de outros objetos
  - Strategy - representação de um algoritmo em um objeto
  - Decorator - atribuição de responsabilidades adicionais a um objeto

Data Access Object (DAO)

Data Access Object (DAO)

Motivação

- A maioria dos sistemas precisa de alguma forma de persistência de dados
- Programadores iniciantes tendem a acessar a base de dados em diversos pontos da aplicação



Solução "tradicional"

Motivação

- Solução "tradicional"
- Vantagens:
  - Performance
  - Mais rápido de implementar (Inicialmente)
- Desvantagens:
  - Dificuldade em realizar modificações (muitas dependências)
  - Repetição de código

Motivação

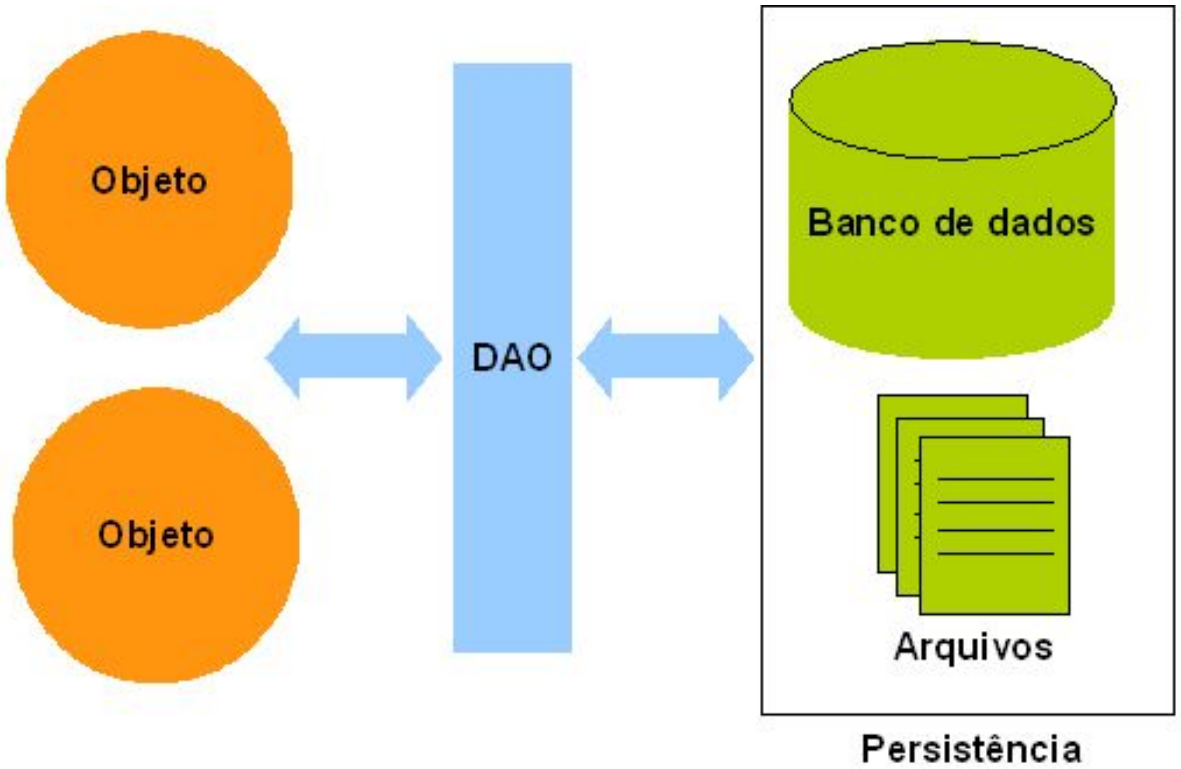
- Diferentes responsabilidades devem ser implementadas em diferentes entidades
  - Interface - Negócio - Persistência (Camadas)
  - Entidade - EntidadeDAO (Data Access Object (DAO))

Descrição

- DAO abstrai o modo de obtenção e gravação dos dados
- Os dados podem ser persistidos de diferentes maneiras
  - banco de dados relacional
  - arquivos XML
  - arquivos
  - ...

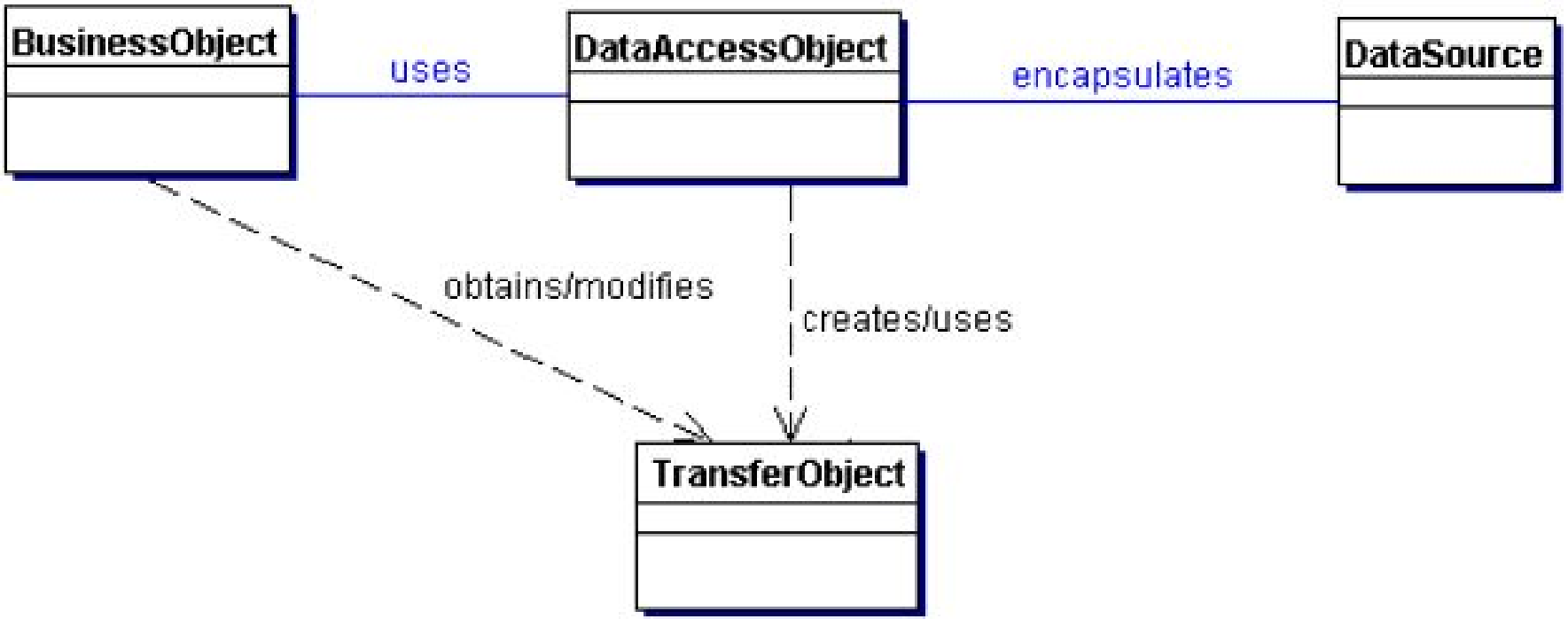
Descrição

- DAO deve funcionar como uma ligação entre a aplicação (camada de negócio) e o meio onde os dados são persistidos



DAO como meio de ligação entre camadas

Estrutura

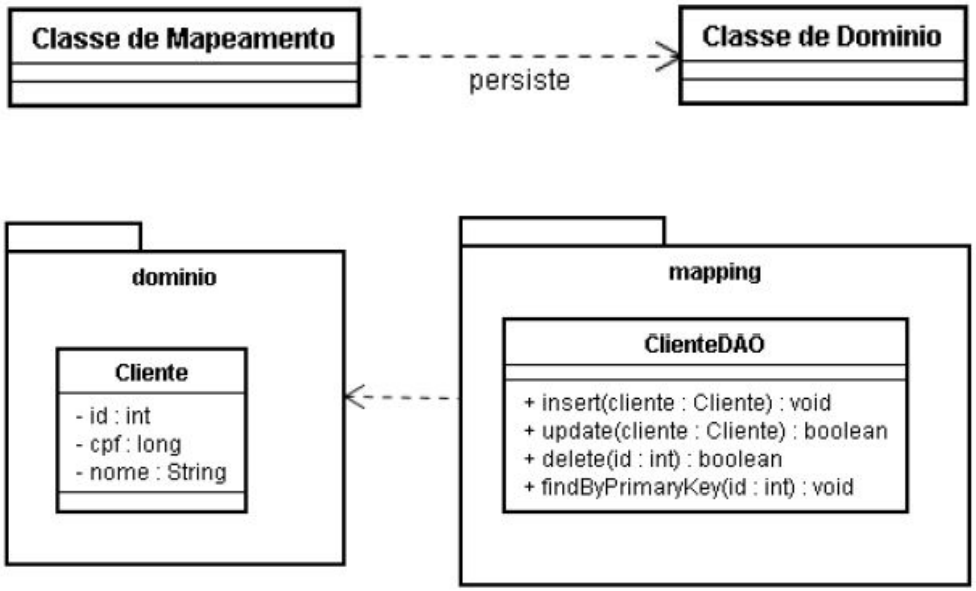


Estrutura do padrão DAO

Participantes

- BusinessObject
  - Implementa a lógica de negócio
- DataAccessObject
  - Realiza a ligação entre a camada de negócio e o meio de persistência
- DataSource
  - Representa o meio de persistência
- TransferObject
  - Representa uma entidade do domínio

Separação de responsabilidades

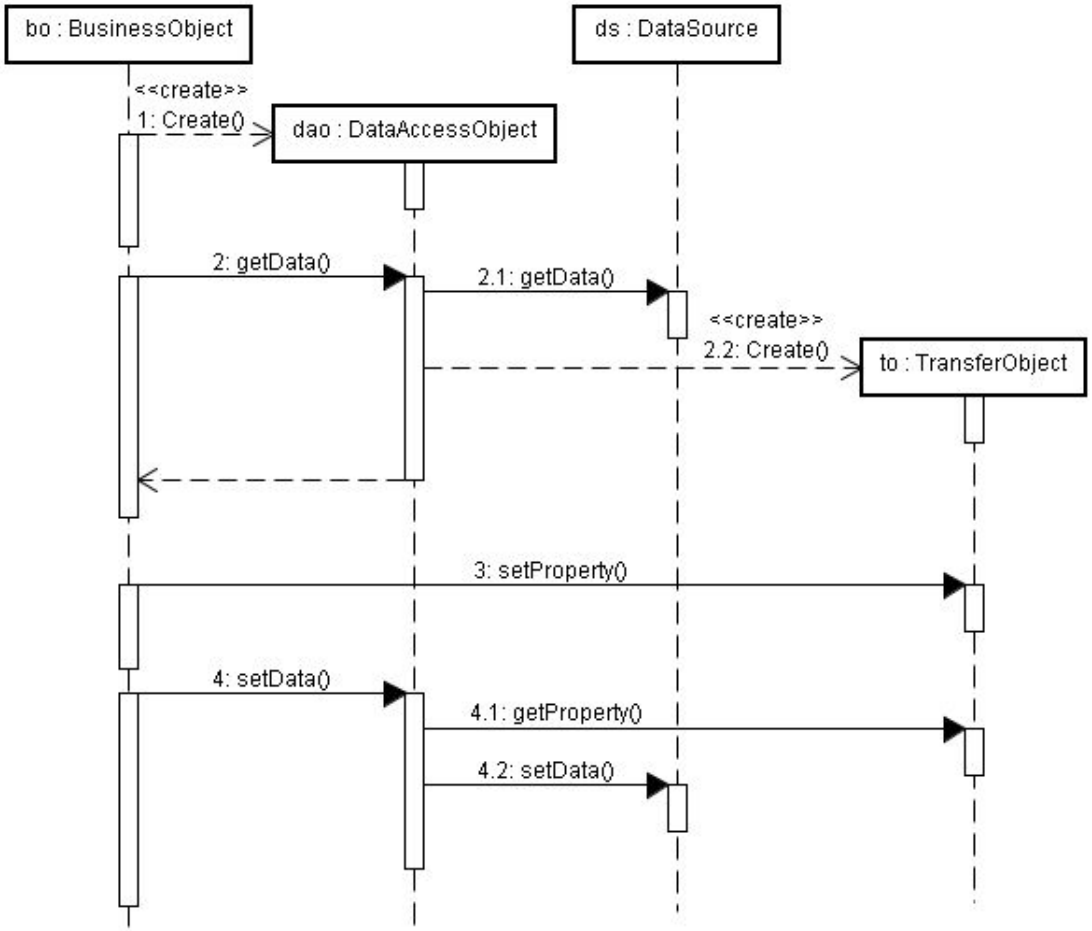


Classes de mapeamento e domÃnio

Colaborações

- BusinessObject Solicita persistência de dados ao DAO
- DAO persiste os dados de TransferObject em um DataSource

Colaborações



Formas de implementação

- DAO é um padrão de projeto (não é um padrão GoF)
- Diferentes formas de implementar o padrão:
  - Criar um DAO para todas as entidades do sistema (somente em sistemas muito simples)
  - Criar um DAO para cada entidade do sistema
  - Criar um DAO para todas as entidades de um módulo
  - Criar um DAO para entidades fortemente relacionadas
  - ...
- Deve-se criar uma interface DAO que será implementada por classes especificas

Exemplos

- Em um sistema de compras existem as seguintes entidades (Classes):
  - Cliente
  - Produto
  - Compra

Exemplos

- Solução 1: Criar um DAO para todas as entidades do sistema (somente em sistemas muito simples)
  - Cliente
  - Produto
  - Compra
  - SistemaDAO

```
public interface SistemaDAOIF {

    public void saveCliente (Cliente c);
    public void deleteCliente (Cliente c);
    public List listClientes ();
    public Cliente findCliente (String name);

    public void saveProduto (Produto p);
    public void deleteProdutoe (Produto p);
    public List listProdutos ();
    public Produto findProduto (String name);

    public void saveCompra (Compra c);
    public void deleteCompra (Compra c);
    public List listCompras ();
    public Compra findCompra (String name);

}
```

```
public class SistemaDAOXML implements SistemaDAOIF {

    public void saveCliente (Cliente c) { ... }
    public void deleteCliente (Cliente c) { ... }
    public List listClientes () { ... }
    public Cliente findCliente (String name) { ... }

    public void saveProduto (Produto p) { ... }
    public void deleteProdutoe (Produto p) { ... }
    public List listProdutos () { ... }
    public Produto findProduto (String name) { ... }

    public void saveCompra (Compra c) { ... }
    public void deleteCompra (Compra c) { ... }
    public List listCompras () { ... }
    public Compra findCompra (String name) { ... }

}
```

```
public class SistemaDAOBDR implements SistemaDAOIF {

    public void saveCliente (Cliente c) { ... }
    public void deleteCliente (Cliente c) { ... }
    public List listClientes () { ... }
    public Cliente findCliente (String name) { ... }

    public void saveProduto (Produto p) { ... }
    public void deleteProdutoe (Produto p) { ... }
    public List listProdutos () { ... }
    public Produto findProduto (String name) { ... }

    public void saveCompra (Compra c) { ... }
    public void deleteCompra (Compra c) { ... }
    public List listCompras () { ... }
    public Compra findCompra (String name) { ... }

}
```

Exemplos

- Solução 2: Criar um DAO para cada entidade do sistema
  - Cliente
  - ClienteDAO
  - Produto
  - ProdutoDAO
  - Compra
  - CompraDAO

```
public interface ClienteDAOIF {
    public void save (Cliente c);
    public void delete (Cliente c);
    public List list ();
    public Cliente find (String name);
}

public class ClienteDAOXML implements ClienteDAOIF {
    public void save (Cliente c) { ... }
    public void delete (Cliente c) { ... }
    public List list () { ... }
    public Cliente find (String name) { ... }
}

public class ClienteDAOBDR implements ClienteDAOIF {
    public void save (Cliente c) { ... }
    public void delete (Cliente c) { ... }
    public List list () { ... }
    public Cliente find (String name) { ... }
}
```

```
public interface ProdutoDAOIF {
    public void save (Produto p);
    public void delete (Produto p);
    public List list ();
    public Produto find (String name);
}

public class ProdutoDAOXML implements ProdutoDAOIF {
    public void save (Produto p) { ... }
    public void delete (Produto p) { ... }
    public List list () { ... }
    public Produto find (String name) { ... }
}

public class ProdutoDAOBDR implements ProdutoDAOIF {
    public void save (Produto p) { ... }
    public void delete (Produto p) { ... }
    public List list () { ... }
    public Produto find (String name) { ... }
}
```

```
public interface CompraDAOIF {
    public void save (Compra c);
    public void delete (Compra c);
    public List list ();
    public Compra find (String name);
}

public class CompraDAOXML implements CompraDAOIF {
    public void save (Compra c) { ... }
    public void delete (Compra c) { ... }
    public List list () { ... }
    public Compra find (String name) { ... }
}

public class CompraDAOBDR implements CompraDAOIF {
    public void save (Compra c) { ... }
    public void delete (Compra c) { ... }
    public List list () { ... }
    public Compra find (String name) { ... }
}
```

Vantagens e Desvantagens

- Vantagens:
  - Classes cliente podem persistir dados sem conhecer os detalhes de cada implementação
  - Mudanças no meio de persistência podem ser realizadas mais facilmente (não será necessário modificar a implementação em diversos pontos da aplicação)
  - Separação de responsabilidades facilita a manutenção e a compreensão
- Desvantagens:
  - Um maior número de classes

Exercício

- Altere o código do projeto 'Agenda', adotando a separação de camadas do MVC
- Complemente o código do projeto 'Agenda', implementando a classe ContatoDAO

Resumo

Resumo

- Model-ViewController - Padrão arquitetura para separação de camadas
- Data Access Object - Não é um padrão GoF - Facilitar o acesso a dados persistentes

Leituras recomendadas

- Código Fonte TV
  MVC // Dicionário do Programador
  `www.youtube.com/watch?v=jyTNhT67ZyY`
- DevMedia
  INtrodução ao padrão MVC
  `www.devmedia.com.br/introducao-ao-padrao-mvc/29308`
- Ram N
  Data Access Object Design Pattern - Introduction
  `www.youtube.com/watch?v=9fVQ_mvzV48`
- Ram N
  Data Access Object Design Pattern - Implementation
  `www.youtube.com/watch?v=H1mePFyqqiE`

Padrão MVC e Padrão DAO

Perguntas?

Alexandre de Andrade Barbosa
alexandre.barbosa@arapiraca.ufal.br